(2)

DECISION
SCIENCE
CONSORTIUM, INC.

TESTING AND EVALUATING C^{3}I SYSTEMS THAT EMPLOY AI

(CLIN 0001)

VOLUME 2: COMPENDIUM OF LESSONS LEARNED FROM TESTING AI SYSTEMS IN THE ARMY

Monica M. Constantine and Jacob W. Ulvila

Decision Science Consortium, Inc.
1895 Preston White Drive, Suite 300
Reston, Virginia 22091

January 1991

Final Report
Period of Performance: 16 September 1988 - 15 September 1990
Contract Number: DAEA18-88-C-0028
PR&C Number: W61DD3-8057-0601
AAP Number: EPG 8048

Prepared for:
U.S. Army Electronic Proving Ground
ATTN: STEEP-ET-S (Mr. Robert J. Harder)
Fort Huachuca, Arizona 85613-7110

TECHNICAL REPORT 90-9

5014

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 90-9		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Decision Science Consortium, Inc.	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION US Army Electronic Proving Ground STEEP-ET-S	
6c. ADDRESS (City, State, and ZIP Code) 1895 Preston White Drive, Suite 300 Reston, Virginia 22091		7b. ADDRESS (City, State, and ZIP Code) Ft. Huachuca, Arizona 85613-7110	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable) STEEP-ET-S	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAEA-18-88-C-0028	
8c. ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS	

PROGRAM ELEMENT NO.	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) TESTING AND EVALUATING C^3I SYSTEMS THAT EMPLOY AI -- VOLUME 2: COMPENDIUM OF LESSONS LEARNED FROM TESTING AI SYSTEMS IN THE ARMY

12. PERSONAL AUTHOR(S) Monica M. Constantine and Jacob W. Ulvila

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical	FROM Sep 88 TO Sep 90	1991, January 31	58

16. SUPPLEMENTARY NOTATION The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Expert Systems, Testing, Knowledge-Based Systems, Artificial Intelligence, Multiattribute Utility

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The field of knowledge-based systems has recently recognized the importance of verification, validation, and testing. This volume presents the results of a survey of the testing practices of knowledge-based systems developers. Common testing strategies are reported and analyzed. Factors affecting testing are discussed. A comprehensive approach to evaluation is described. General conclusions and lessons learned are presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Mr. Robert J. Harder	22b. TELEPHONE (Include Area Code) (602) 538-2090 — 22c. OFFICE SYMBOL STEEP-ET-S

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED

CONTENTS

CONTENTS (Continued)

1.0 CVERVIEW

This volume presents the results of the compendium of lessons learned performed on the Phase 2 small business innovative research (SBIR) project on "Specifying, Testing, and Evaluating Systems that Employ AI." The objective of the compendium of lessons learned was to determine the state of the practice for testing knowledge-based systems in the Army and to assess the impact of those experiences on methods and procedures for testing knowledge-based systems. The compendium of lessons learned resulted in two articles: one for the IJCAI-89 Workshop on Verification, Validation, and Testing of Knowledge Based Systems, and one that was more generalized for publication in *Expert Systems with Applications: An International Journal*. The first article presented the findings of the interviews conducted and summarized the lessons learned. The second article generalized the findings for a broader audience. The articles are found in Volume 4 of the final report.

This volume serves to describe in more detail the interviews and presents the general findings based on these interviews. Section 2.0 describes interviews and Section 3.0 presents the generalized findings and lessons learned.

2.0 DATA COLLECTION

THE INTERVIEWS

Potential candidates were selected from various AI Activity Point of Contact lists Generally, information on contacts came from one of three sources: the points-of-contact for AI activities published in the *AI Exchange* newsletter prepared by the U.S. Military Academy Office of AI Analysis and Evaluation, the list of Artificial Intelligence Projects, FY90-91 prepared by the Office of the Deputy Chief of Staff for Logistics, and from information obtained at the AI proponency conference held in the summer of 1989.

The interviews were primarily conducted with Army representatives local to the Washington D.C. area. Additionally, telephone interviews were conducted with representatives that were out of state in order to obtain representation from the various major commands. Generally, the in-person interviews were more informative than the telephone interviews. The interviews conducted in person usually took between 1 and 2 hours. In all cases of the in-person interviews, the participants were very cooperative and willing to share information. The telephone interviews were not as successful; it was much easier for the interviewee to give short quick answers rather than discuss the problems faced in testing expert systems. (Appendix A to this volume contains the points-of-contact used for the compendium.)

The interviews were relatively open-ended. A list of issues was developed prior to the interview, and the interviewer used the list to direct the interview and took notes on the discussion. The role of the interviewer was basically one of listener while the interviewee did most of the talking. The questions and issues for the interviews are listed in Table 1. The list includes questions relating to a general description of the system, why the system was considered AI, requirements for the system, the type of testing performed, the properties of the system tested, the techniques used for testing, testing tools, and general comments and advice to others.

Table 1: Compendium of Lessons Learned: Questions and Issues

System Description

1) The size of the program as well as databases.
2) The criticality of the system.
3) The required software reliability.
4) The critical or unique aspects of the system.
5) The stage in the life cycle.
6) The importance of execution time and time to solution.
7) The machine used and storage constraints.
8) The intended user of the system, whether or not a "human" equivalent to the system exists, the user's interaction with the system, and the setting in which the software is designed to be used.
9) The language used to develop the system. Is the language used to develop the prototype different from the language used to develop the operational system?
10) The maintenance of the system. Does the end user have the ability to modify the rule base?

Other System Issues

1) The use of software tools and shells.
2) The formality of the development environment.
3) The software developer. Was the system developed by the end user, expert, or software group? Was the system developed in-house or by a contractor.

Why is the System Considered AI

1) Is the system problem-oriented? Does it solve a problem previously performed using human judgement and experience?
2) Is the system technique-oriented? Can the system be identified with an AI programming strategy? Is it based on heuristics and representations?
3) Does the system use deep knowledge? Does the system model particular world principles or use axioms and laws to make inferences and deductions?
4) Does the system use surface knowledge--rules-of-thumb human experts commonly deploy?
5) What defines an "expert?"

Requirements for the System

1) Were requirements generated formally or informally?
2) Does documentation exist? If not, how are requirements defined and communicated?
3) How frequently do the requirements change?
4) Was a prototype used to help define requirements?

Table 1: Compendium of Lessons Learned: Questions and Issues (continued)

Pieces of the Software Tested

1) The inference engine, logic scheme, or algorithms.
2) The knowledge base.
3) The user interface.
4) The interaction between the man and the machine.
5) The model (especially important if using deep knowledge).

Type of Testing

1) Is each piece of the software tested?
2) Is the software tested against some standard? What is the standard; how was the standard defined?
3) Does "ground truth" exist?
4) How was the expert's opinion used in the testing process?
5) Were single or multiple experts used to test the software?
6) Was the software tested against requirements?
7) Was the software tested with the end user?
8) Is testing integrated with the life cycle of the expert system?
9) Is more emphasis placed in one phase of the life cycle than another?

Properties Tested

1) System effectiveness.
2) Quality and success rate.
3) Timeliness of responses.
4) Accuracy.
5) Bias.
6) Error handling.
7) Organizational effectiveness (how well does the software fit with the intended user and the work environment?).

Testing Techniques

1) Static testing versus dynamic testing.
2) Use of automated testing tools.
3) Use of a controlled experiment.
4) Test case selection. Determining a comprehensive set of test cases.
5) Use of actual data, use of simulated data in testing.
6) Stress testing.
7) Testing the knowledge base. Does the system know what it does not know?
8) User testing——use of a novice or expert; single or group of users.
9) Were the same test cases used in development and testing?
10) Were the same experts used in design and testing?

Table 1: Compendium of Lessons Learned: Questions and Issues (continued)

Testing Tools

1) Was a testing tool used?
2) What features of the testing tool were used?
3) Were the tools easy to use?
4) Did using the tools help isolate faults or failures in software that made their use justified?
5) Were metrics used? How were the metrics interpreted?
6) When does testing stop?
7) What features of the tool would be most beneficial?
8) Were testing tools tried, but not used?
9) What kind of testing tools would make the job of a software tester easier?

Other

1) Was testing formally in the process or was it an afterthought?
2) Were any standards (formal or informal) used for languages and/or shells?
3) Was uncertainty used? Which form?
4) What was the development model?
5) What was the greatest difficulty or biggest stumbling block?
6) Can you provide any advice for others testing a similar system?

SUMMARY OF INTERVIEWS

The interviews indicated the diversity of expert system applications in the Army. Knowledge-based systems have been developed for battle management, hazardous material classification, contract clause selection, in-house support for medical research (vaccine testing and immunization scheduling), and assigning a benefit rating for psychiatric disability compensation. The interviews presented in Appendix B represent the variety of expert system applications and testing methods. Basically, the variety of testing approaches were classified into eight commonly employed strategies: prototype forever, agreement, compliance, satisfaction, case-dependent, organizational testing, field testing, and multi faceted.

(1) *Prototype Forever*. The expert receives the latest version of the software and uses it in an actual setting. The expert monitors the system in use and provides feedback on the interface, the explanation facility, and the reasonableness of the system's outputs over time.

(2) *Agreement*. As the system is being developed, it is tested with the expert. When an initial version of the system is complete, a sample of test cases is selected based on actual data. The test cases are given to an expert or a panel of experts who is asked to determine the outcome. The same set of test cases is presented to the system and the system determines the outcome. The system passes the "test" if the system and panel of experts agree on the outcome for some percentage (e.g., 85%) of the test cases. The system is put into use and monitored over time.

(3) *Compliance*. Test cases are selected based on past history. Those cases are presented to the system, the system's performance is compared with the historical results, and appropriate changes are made. Another set of test cases is selected from current data where the outcome is not yet known. These cases are presented to the system and the output is correct if it complies with the relevant regulations.

(4) *Satisfaction*. The developer examines the knowledge base for missing rules, rules that can be collapsed, and rules that are not being fired. The expert subjectively assesses the correctness of the rules, the quality of the explanations, and the quality of the answers. The user assesses his or her ability to interface with the system, the timeliness of the response, the reasonableness of the outputs and explanations, and how the system fits in with the operating environment.

(5) *Case-Dependent*. The developer examines the knowledge base, assesses the effect of adding rules, determines if rules can be

ombined, and looks for errors. A large sample of test cases is selected that approximates the population of cases the system will receive. The expert assesses the answer to test cases without using the system. Then, the expert uses the system to obtain outputs (all of the expert's actions are saved). The saved data reflecting the expert's actions are analyzed and changes are made to the knowledge base. It is necessary for the expert and system to agree some percentage (e.g., 80%) of the time. The system is then tested with the non-expert users. The non-experts interpret the input data from summary sheets and the differences in data input between the expert and non-expert are examined and appropriate changes are made to the system.

(6) *Organizational Testing.* The interface is iteratively evaluated by the user. Interface evaluation includes an assessment of screen design, feedback message placement, scrolling, features, menu naming, design, and actions. The system is evaluated in a classroom setting by observing the system in use and administering questionnaires. Observers videotape and take notes to assess how both students and instructors use the system in an actual classroom setting. Questionnaires are administered to both students and instructors to gather information regarding features used, perceived usefulness, perceived problems, and general feelings. An experiment using subjects in an actual classroom environment is designed to assess the effect of using the system on student performance.

(7) *Field Testing.* Each prototype is tested with past cases from saved actual data. The system is tested in a similar operational environment for several (e.g., 3) months to obtain feedback on system effectiveness and user interface. Appropriate changes are made to the system. The system is then run in parallel with the existing process in the intended operational environment for approximately one year. During the parallel test, assessments are made as to how well the system is meeting the goals stated in the requirements document.

(8) *Multi-Faceted.* The developer performs a comprehensive static analysis of the knowledge base using automated tools. Dynamic testing is performed to test the system with the expert using a comprehensive set of test cases not used in development. Multi-attribute analysis is used to obtain subjective measures for system performance. The system is tested with "developer" experts as well as outside experts. Questionnaires are administered to both developer experts and outside experts.

Despite the diversity of systems represented in the interviews and the different approaches to testing, the systems were all quite similar in their goals. Most functioned as an aid to a decision maker rather than as a decision maker. Generally, the systems reduced the amount of time it takes to perform a specific task or added consistency to a task that is performed differently by different individuals. Although the size of the system's rule bases varied, most contained information from regulations as well as subject-area experts.

Most developers worked in a relatively informal development environment without documented requirements. The system requirements evolved from an in-house need or from an existing program. The smaller systems were generally developed using an expert system shell, while the larger systems were generally developed with a lower-level computer language such as LISP or C. Most of the systems were developed as prototypes and used rapid prototyping as a development methodology. The lack of information on test methodologies, the lack of documented requirements, and the 'fuzzy' nature of knowledge-based systems generally made testing difficult. Generally, time, resources, and the lack of information had a large impact upon the amount of testing performed.

SURVEY OF ARTIFICIAL INTELLIGENCE IN THE ARMY - SUMMARY

In addition to the interviews, a survey of AI in the Army was distributed at the May 1990 Army AI Proponency Conference. The purpose of the survey was to update some of the information obtained in the interviews and to reach organizations that were not included in the initial compendium of lessons learned. The survey also verified some of the conclusions drawn from the compendium of lessons learned. The survey asked questions pertaining to the level of effort expended, the primary mission of an AI system, testing strategies, factors that affect testing, the aspects of AI systems that were tested, and the importance of traditional quality factors for an AI system. The survey also asked the respondents to identify areas in which future woik is needed. This section presents the general results of the survey. Appendix C contains a copy of the survey used an a tabulation of the results.

- *Level of Effort*. The majority of AI systems were small, requiring fewer than 24 man-months of effort. Only one office was developing a very large system that required more than 60 man-months.
- *Development Environment*. The most common platform for AI systems development seems to be microcomputers and workstations. Seven of the twelve respondents had at least some systems developed with an AI shell and seven of the respondents had at least some systems developed using an AI language such as Common LISP. The most

common kind of AI systems developed were expert systems. Few offices created intelligent tutoring, natural language, or neural network systems. Only one office was developing a system that was classified as an image recognition system.

- *System Classification.* The survey asked the respondents to classify the systems as mission critical, assisting an expert, assisting a novice, and as systems that operate autonomously. Four out of twelve offices surveyed said that they were developing at least one mission critical system and two offices said they were developing a system that was designed to operate autonomously. However, the majority of systems were designed to assist either an expert or novice in solving a problem.

- *System Distribution.* Three offices were creating systems whose users are outside the developing organization. However, most organizations developing systems are users as well as developers. All of the offices surveyed were developing at least one system that would be widely distributed within the Army.

- *Testing Strategies.* As a result of the interviews, eight common testing strategies were identified. The respondents were asked if any of their AI systems were tested with similar approaches. Prototype forever was listed as the most commonly used approach, followed by agreement, field testing, and satisfaction. Case-dependent and organizational testing were the least commonly applied strategies. Two organizations represented by the survey used a multi-faceted testing approach on at least some of the AI systems developed.

- *Factors that Affect Testing.* Based on the interviews, four factors were found to affect the amount of testing performed on an AI system—time, resources, information available on testing, and the development environment. The respondents were asked to rank the factors on a scale of one to five, where five represented a large impact. Overall, time and resource constraints were listed as the factors having the most impact on testing. Both information available on testing and the development environment were rated as important, but to a lesser degree than time and resources.

- *Aspects of an Expert System.* The respondents were asked how often and how important the different aspects of an expert system were in testing. The survey used the aspects addressed in the MAU hierarchy, the structure and content of the knowledge base, the inference engine, the service requirements, performance, and usability. Performance, usability, and the structure and content of the knowledge base were listed as the aspects that were tested most often, as well as the aspects that were most important. Testing service requirements and the inference engine were viewed as less important and were also aspects that received the least amount of testing.

- *Software Quality Factors*. The survey listed 11 quality factors used in traditional software testing and the respondents were asked to rank the importance (high, medium, and low) of these factors in testing AI software. On the average, none of the software quality factors received a low ranking. Usability, reliability, and correctness were ranked as the most important factors, followed by testability, flexibility, interoperability, portability, maintainability, integrity, and reusability.

- *Future Work*. The survey listed 11 possible areas for future work and asked the respondents to rank (high, medium, or low) the importance of future work. The following list shows the subject areas, as well as the average ranking received:

  - *High*: Develop automated static testing tools for analyzing the consistency and completeness of knowledge bases.

  - *Medium*: Develop methods and aids for determining the accuracy and built-in bias of an expert system. Periodically update and publish a compendium of lessons learned from efforts to test Army AI systems. Collect and publish an anthology of articles on testing AI.

  - *Medium-Low*: Develop methods for testing the validity of "confidence" or "uncertainty" factors which are used in expert systems. Develop an automated means for assessing user acceptance. Develop a tool to generate requirements against which an AI system can be tested. Develop a computer program to help perform multi-faceted testing. Try out AI testing methods in a testbed. (Only one office indicated that it may be able to serve as a possible testbed.)

3.0 GENERAL FINDINGS AND LESSONS LEARNED

This section lists characteristics of the typical development environment and characterizes the eight testing strategies in terms of the development environment and the MAU hierarchy shown in Figure 1. Volume 1 contains more detailed information of the MAU hierarchy. The section concludes by summarizing some of the lessons learned, particularly as they relate to testing. (Please refer to Volume 4 of the final report for additional information on the lessons learned.)

In addition, the lessons learned were influenced by our attempts to apply the extensive testing methods described in Volume 1 to several expert systems. These systems included an intelligence expert system, a railroad box car assignment expert system, and a vehicle test equipment configuration expert system. For reasons of timing and resource constraints, these attempts were not totally successful and were not completed.

CHARACTERIZATION OF THE EXPERT SYSTEM DEVELOPMENT ENVIRONMENT

This section presents a summary of the development environment, based on information obtained in the interviews, and the survey.

- *Hardware.* Typically, the systems were designed for use on PCs and workstations. However, some systems were developed for mainframes, Symbolics, and SUNs.
- *Does the System Need to be Ported to Another Machine?* About half the systems will need to be ported to a different machine before they can become operational.
- *Is the Machine Operating Near its Limits?* In most cases, the machine is not operating near its limits.
- *Was a Shell Used?* Generally, developers in the Army use expert system shells. The various shells used were: M1, KREME (object-oriented), FLAVORS (object-oriented), KEE, LEVEL5, 1st Class, XPER, XSYS, CLIPS, and ART. A smaller number of systems are developed with an AI language such as Common LISP.
- *Other Development Tools Used.* Typically, other development tools were not used. In one case, a system included a Hypertext module.
- *Static Analysis Tools.* Typically, these were not used because they were not available.

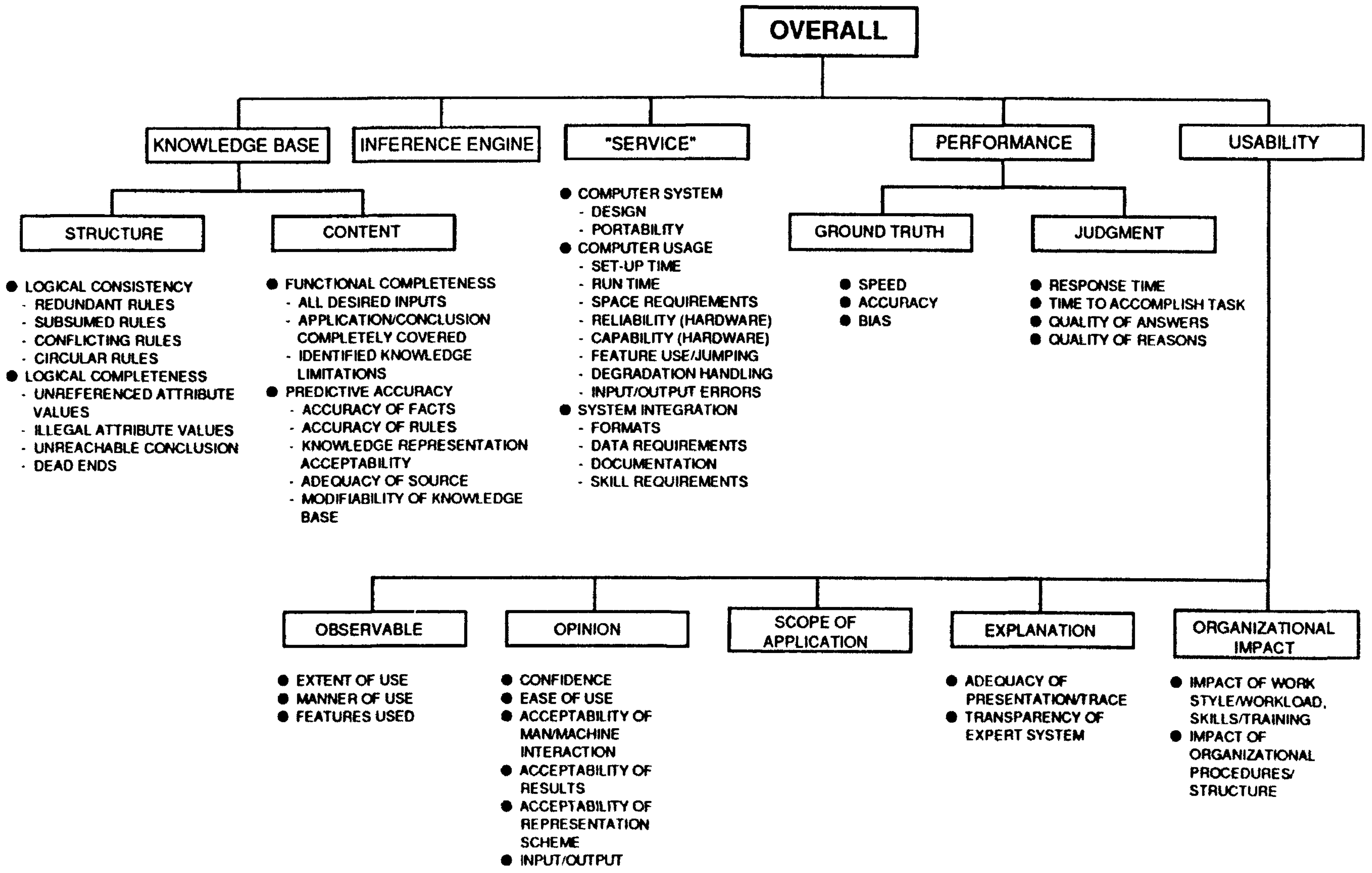

Figure 1: A MAU Framework for Integrating Test and Evaluation Criteria

- *Access to Other Systems.* Typically, the systems did not need to access or interface with other computer systems.

- *Access to Databases.* Typically, systems did not have to access other existing databases. However, there were cases where access to a large database was a critical aspect of the system.

- *Access to Material about the Domain.* In almost all cases, there was some limited amount of information regarding the domain (typically in a regulation).

- *Is there a Recognized Authority in the Domain Area of the System?* Typically, there was no one recognized authority.

- *Documented Standards.* Typically, there was no documentation on expected performance, input domain, or scope of application.

- *Documented Test Plan.* Typically, there was no documented test plan.

- *Primary Mission.* Most systems were primarily designed to assist either an expert or novice in solving a problem. There are a few systems that would be classified as mission critical and even one system that was designed to operate autonomously.

- *Most Critical Part of the System.* In most cases, the most critical part of the system was the knowledge base.

- *Intended User.* The systems were designed for use by both experts and nonexperts. Typically, the developing organization was also a user of the system.

- *Distribution.* Typically, systems are distributed widely within an organization. However, there are a few systems developed that will be widely distributed within the Army.

- *Early Involvement with Intended Users.* When the intended users are not the same as the experts, the intended users were not involved early in the development or testing process. (In some cases, the intended users were not involved in either testing or development.)

- *Availability of Experts.* Typically, it is very difficult to get the expert's time.

- *Availability of Users.* When the intended users are a different group from the experts, they generally have not been considered in the development and testing process. For some systems, accessing typical users would be a possibility.

- *Test Cases.* Typically, a limited number of test cases are available.

- *Same Test Cases used in Development and Testing.* Typically, some test cases are withheld and not used in development. However, some systems have used the same set of test cases for both testing and development.

- *Mission Criticality.* Only a few systems were classified as mission-critical.

- *Developed as Prototype.* All of the systems were developed iteratively as prototypes.

- *Is the System Operational?* There are plans for about half of the systems to become operational after further development. A few are currently operational.

- *Who is the Developer?* Most systems were developed by contractors; others were developed by in-house Army personnel.

- *Resources.* Generally resources are tightly constrained.

- *Is the System Tested Iteratively?.* Almost all of the systems were tested iteratively throughout the development process and did not have a formal test phase.

- *What Part of the System was Emphasized during Development, Testing?* The survey indicated that the knowledge base, performance, and usability were aspects that were both important and frequently tested. However, in the interviews, it seemed that the knowledge base was emphasized during development, and the correctness or reasonableness of the answers (as judged by the expert) was emphasized during testing.

CHARACTERIZATION OF COMMON TESTING STRATEGIES IN TERMS OF THE ENVIRONMENT

Table 2 characterizes the eight testing strategies in terms of the development environment. All of the testing strategies except organizational testing, field testing, and multi-faceted testing were used in informal development environments where there were no documented requirements or test plan. The prototype forever, agreement, and compliance strategies were typically used when the expert system was designed for a small number of in-house users. Even though the expert's time was generally very difficult to obtain, all of the testing strategies (except compliance) used the expert in the testing process.

Table 2: Characterization of Testing Strategies in Terms of Development Environment

CHARACTERISTICS \ TESTING STRATEGIES	PROTOTYPE FOREVER	AGREEMENT	COMPLIANCE	SATISFACTION	CASE-DEPENDENT	ORGANIZATIONAL TESTING	FIELD TESTING	MULTI-FACETED
1. FORMAL DEVELOPMENT ENVIRONMENT	NO	NO	NO	NO	NO	YES	YES	YES
2. DOCUMENTED REQUIREMENTS	NO	NO	NO	NO	NO	YES	YES	YES
3. DOCUMENTED TEST PLAN	NO	NO	NO	NO	NO	YES	YES	YES
4. EXPERT IS USER	YES	YES	YES	YES	NO	NO	YES	YES
5. NON-EXPERT IS USER	NO	YES	YES	NO	YES	YES	NO	NO
6. LIMITED NUMBER OF IN-HOUSE USERS	YES	YES	YES	NO	NO	NO	NO	NO
7. POTENTIAL FOR DISTRIBUTION TO A LARGE NUMBER OF USERS	NO	NO	NO	YES	YES	YES	YES	YES
8. COST OF ERROR IS LOW	YES	YES	YES	YES	YES	YES	YES	NO
9. SYSTEM PERFORMS A SUPPORT FUNCTION	YES	YES	YES	YES	YES	YES	YES	YES
10. SYSTEM FUNCTIONS AS A DECISION MAKER	NO	NO	NO	NO	NO	NO	NO	NO
11. SYSTEM IS PRIMARILY AN AID TO A DECISION MAKER	NO	YES	NO	YES	YES	NO	YES	YES
12. SYSTEM IS PRIMARILY DESIGNED TO ADD CONSISTENCY TO AN EXISTING PROCESS	NO	NO	YES	NO	YES	NO	NO	NO
13. SYSTEM IS PRIMARILY DESIGNED TO SAVE TIME & MONEY	NO	NO	NO	NO	NO	YES	YES	NO
14. EMBEDDED KNOWLEDGE IS FROM REGULATIONS	YES	YES	YES	YES	NO	YES	YES	YES
15. EMBEDDED KNOWLEDGE IS FROM SINGLE EXPERT	YES	YES	NO	NO	YES	NO	NO	NO
16. EMBEDDED KNOWLEDGE IS FROM MULTIPLE EXPERTS	NO	NO	NO	YES	NO	YES	YES	YES
17. CLOSE WORKING RELATIONSHIP BETWEEN DEVELOPER & USER	YES	NO	NO	NO	NO	YES	NO	NO
18. EXPERT WAS AVAILABLE FOR SOME DEVELOPMENT & TESTING	YES	YES	NO	YES	YES	YES	YES	YES

CHARACTERIZATION OF COMMON TESTING STRATEGIES IN TERMS OF THE MAU HIERARCHY

Figures 2 to 5 characterize each of the testing strategies according to the MAU hierarchy. The shaded items on the hierarchy indicate whether the attribute was addressed by a particular testing strategy. At some level, all of the common testing strategies addressed correctness of the answer and correctness of the reasoning. All of the systems were judged against some standard, although in many cases that standard was simply a vague notion of "correctness." For example, in agreement testing, the system's output was stamped "correct" if it agreed with the outcomes specified by a panel of experts for 85% of the test cases. In case-dependent testing, great care was taken to select a set of test cases that approximated the kinds of cases the system would be expected to handle in an operational environment, but all of the cases used in testing were also used in development. In field testing, the system was judged "correct" if it met a few performance criteria (such as a 30% reduction in downtime) specified in a requirements document.

In one particular project—an intelligent tutoring system—correctness was not as important as clarity of reasoning, usability, or how well the system fit into the intended operational environment. These testers used organizational testing because the other testing strategies did not sufficiently address either usability or fit with the organization.

Many of the testing strategies (prototype forever, agreement, compliance, and field testing) did not directly address either the structure or content of the knowledge base. The satisfaction and case-dependent strategies insufficiently addressed testing the knowledge base due to the lack of automated tools for static analysis. Most of the developers realized the importance of structural or static testing but, without automated tools, lacked the resources or time to do as much testing as they would have liked. Only the multi-faceted testing strategy used an automated static analysis tool, and it was developed especially for the particular application.

Most of the knowledge-based systems made use of one of the many expert system shells available on the market, and no testing was specifically aimed at the inference engine of the shell. Most developers assumed that, when they purchased a shell, the inference engine had already been tested thoroughly.

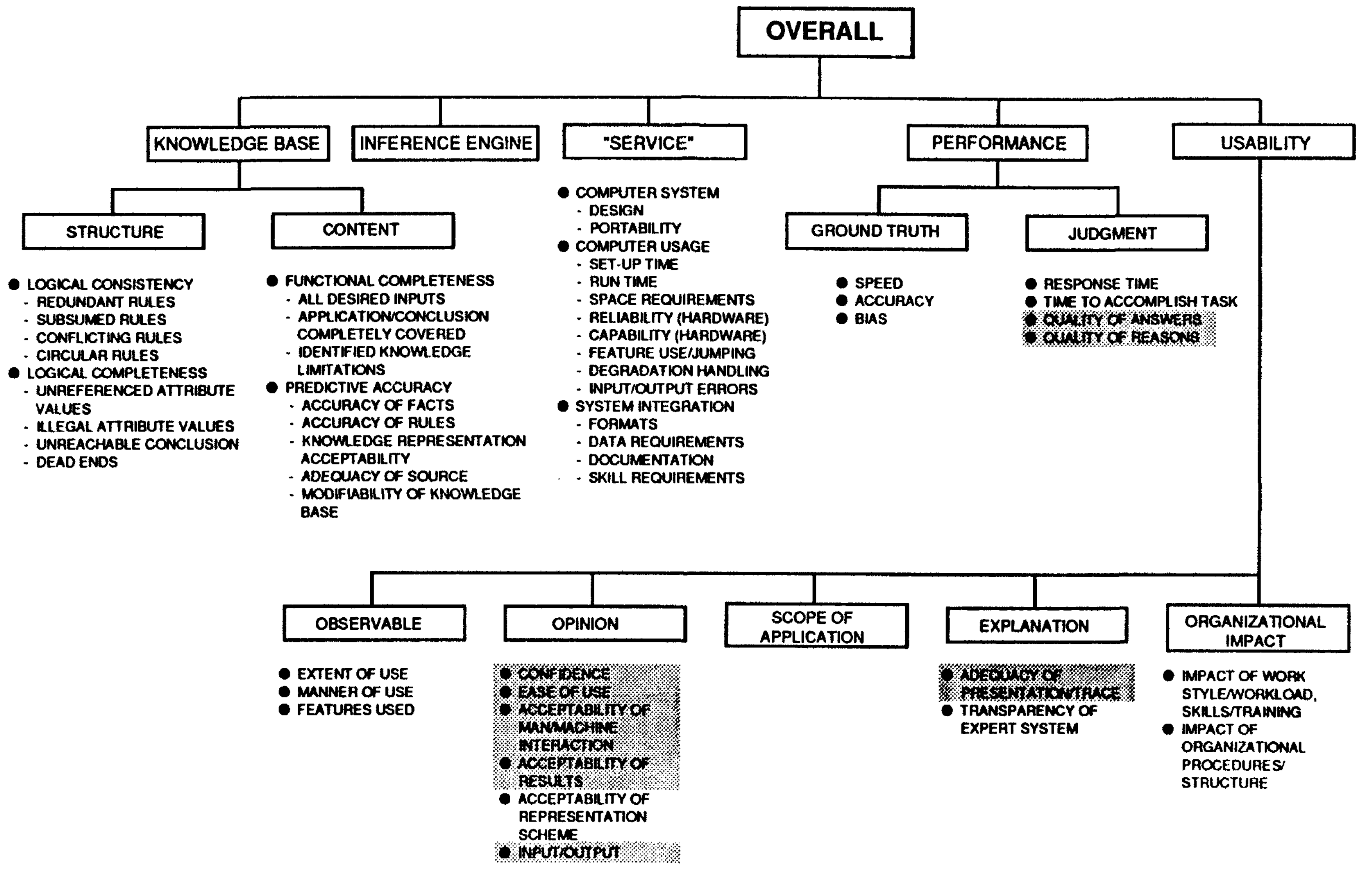

Figure 2: Prototype Forever, Agreement, Compliance, and Field Testing Strategies Characterized in Terms of MAU Framework

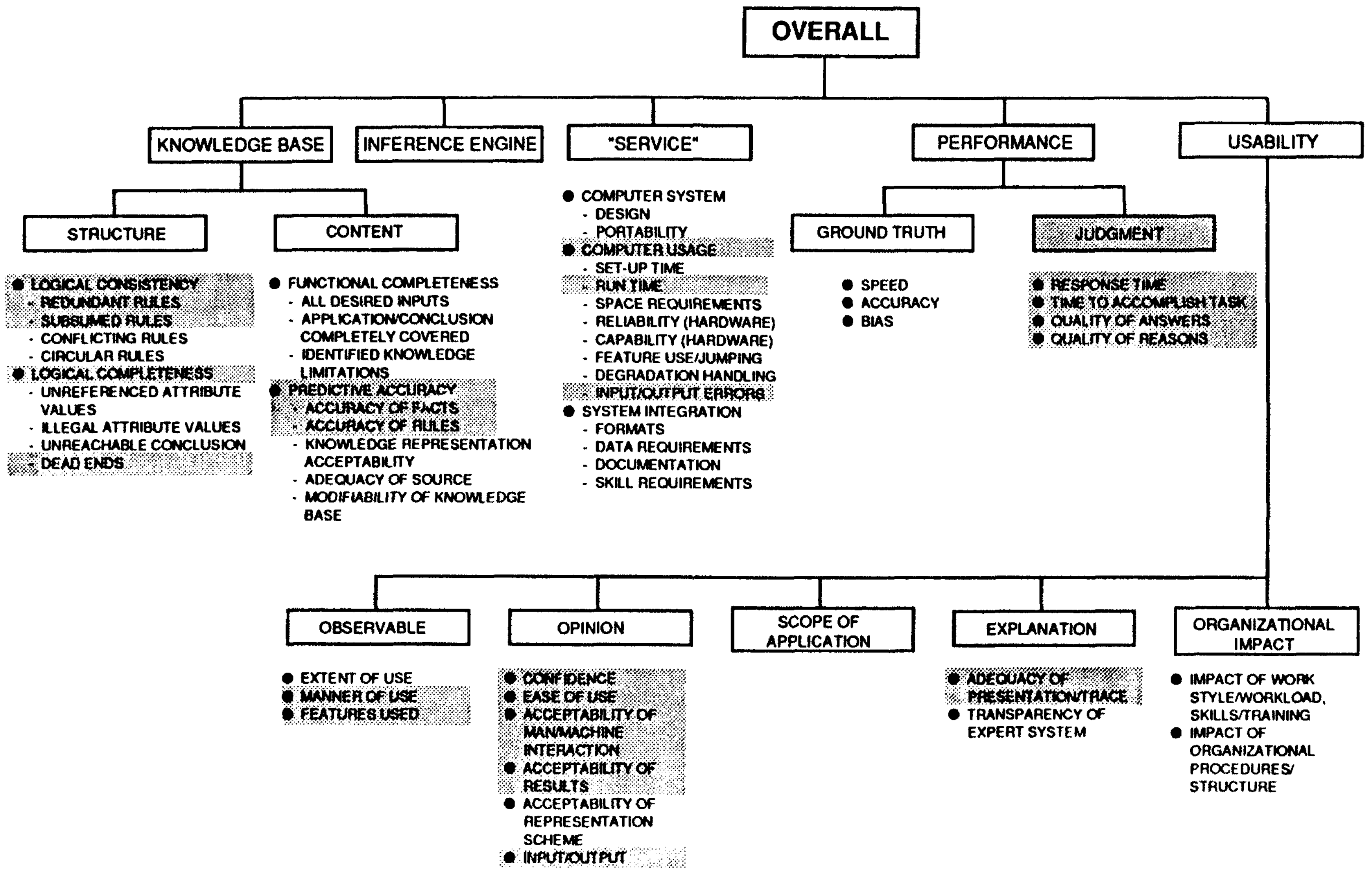

Figure 3: Satisfaction and Case-Dependent Strategies Characterized in Terms of MAU Framework

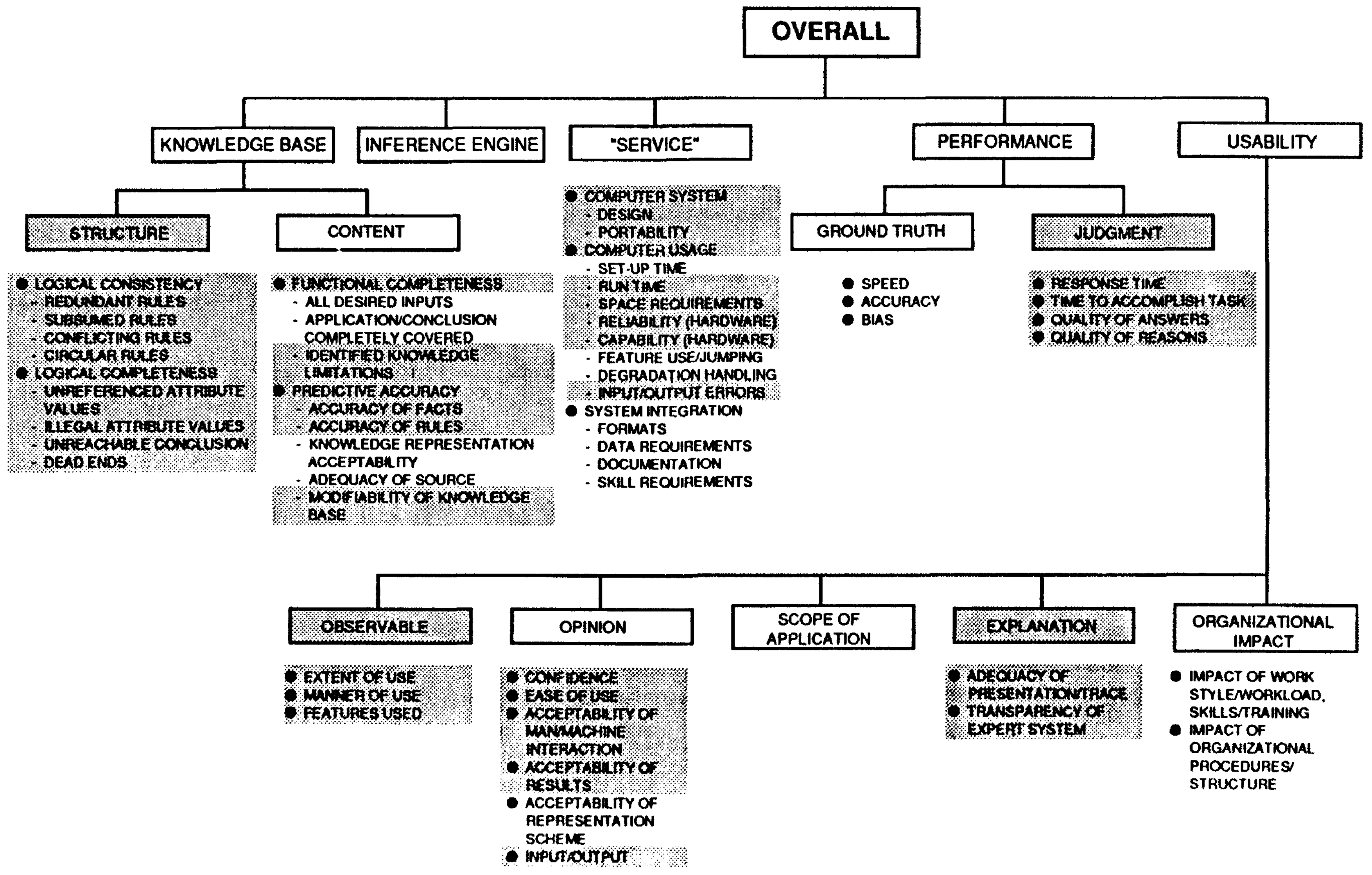

Figure 4: Multi-Faceted Strategy Characterized in Terms of MAU Framework

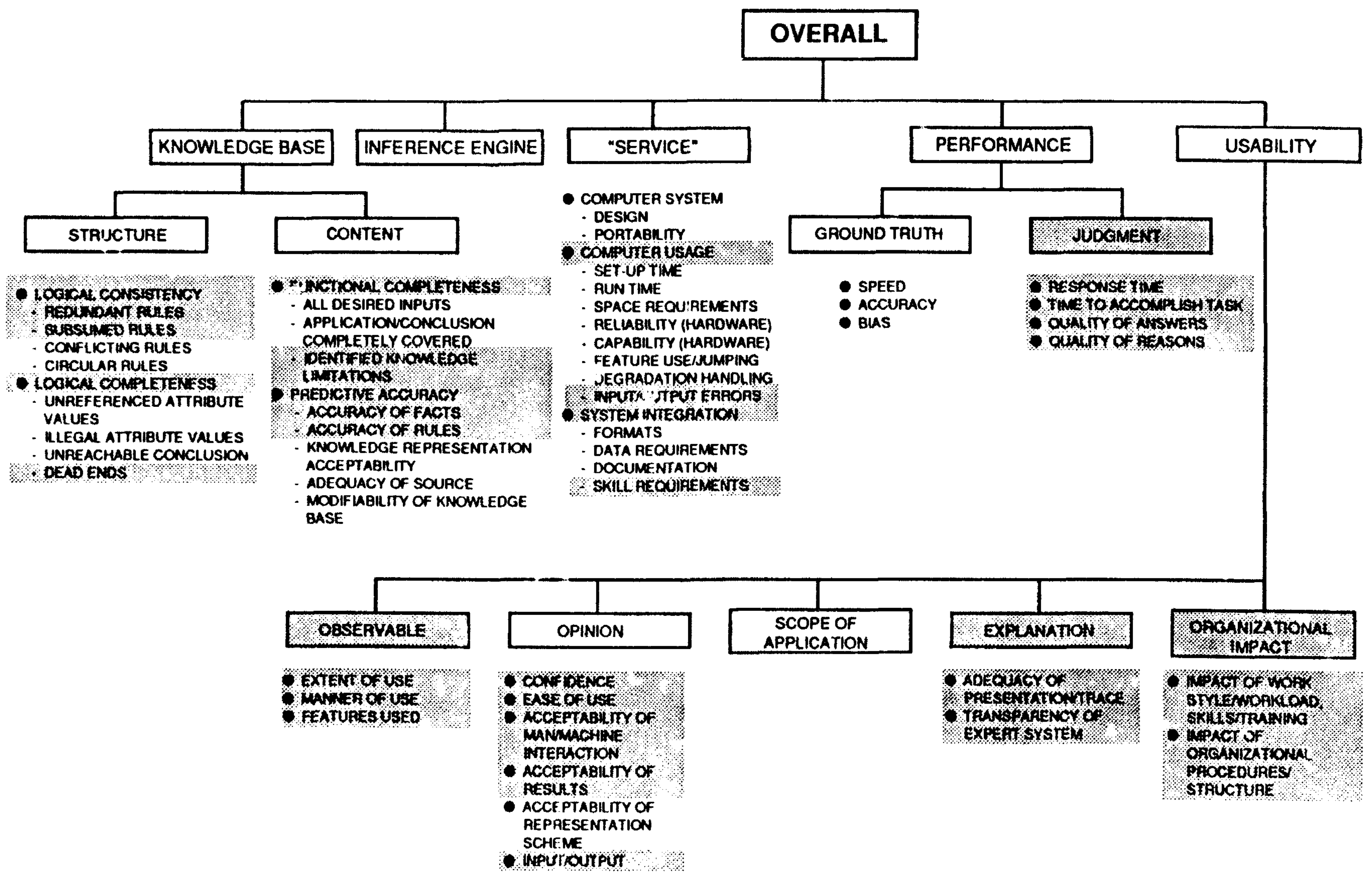

Figure 5: Organizational Testing Strategy Characterized in Terms of MAU Framework

This may not be the case. The literature indicates that perhaps only one inference engine—CLIPS—has been formally validated.

The testing strategy of changing inputs, obtaining outputs, and asking the expert if the results are reasonable may be appropriate for small, expendable, non-autonomous, non-critical, in-house systems (where the cost of an error and the cost of the system are extremely low), in environments where the developer and the expert work closely together and the performance of the system is continually monitored. But be prepared; this system is likely to remain a prototype forever. When a knowledge-based system is to be used by a large number of individuals, replace an existing method for solving a particular problem, perform an important or critical function, or where the cost of a system error may be high, more rigorous and thorough testing is necessary.

CONSTRAINTS TO TESTING

This section outlines some of the constraints a tester may encounter and outlines the implications.

Hardware Constraints

- The system is operating near machine limits.
- The system needs to be ported to another machine.

Testing Implications: The tester must be sure to have access to other computers on which the system is designed to run. If the machine is operating near limits, the tester should determine if there are conditions that cause the system to run at an unacceptable level.

Interoperability Constraints

- The system must access external databases or must access other computer systems.

Testing Implications: The tester needs to ensure access to the other databases or systems for testing. This may mean scheduling some downtime for the users of these databases or systems.

Data Constraints

- Access to materials about the domain;
- Access to test cases;
- All of the test cases used in development;
- Expected input domain not defined.

Testing Implications: Most of the systems discussed in the interviews had some type of data to test the system. If actual data do not exist or all of the test cases were used in development, testers need to work with developers and experts to develop a new set of test cases or to use data that resulted from a simulation.

If actual data exist, it may be useful to analyze the data to get information on the average case the system would be expected to handle, the range of these cases, as well as types of cases the system would not handle. It may be useful to meet with the developers to assess the input domain (the distribution and critical values).

If no test cases are available, the best the tester could do is test the other aspects of the system, such as the knowledge base.

Tool Constraints

- No access to static analysis tools;
- No access to statistical packages;
- No access to questionnaires.

Testing Implications: Typically, static analysis tools, statistical packages, and questionnaires were not used or available. Without tools for static analysis, extensive manual analysis of a large rule base is practically impossible. The tester may try to divide a large rule base into logical subsets of rules and test the subsets for logical consistency and logical completeness. If the rule base is small (fewer than 50 rules), the tester may manually examine the rule base for logical consistency and logical completeness.

Documentation Constraints

- No objectives for testing (test plan);
- No document requirements;
- No standards against which to test;
- No explicitly defined performance measures.

Testing Implications: Documentation rarely exists and if it does, it usually focuses on hardware requirements and issues about system integration. Measures of performance are not usually explicitly defined, but probably exist. The tester should meet with the developer to make the implicit performance requirements explicit and document these requirements; otherwise the tester will not be able to make any assessment regarding the system's performance.

Personnel Constraints

- Access to experts;
- Access to intended users.

Testing Implications: Typically, when the expert is not available to test the software, the project fails. If the expert is the intended user, or a suitable replacement does not exist, the tester basically has two choices—to either act as the expert or intended user himself, or to test only those aspects of the system that do not include the judgment or opinion of the user or expert. The tester may simply run test cases through the system and check for cases that cause the system to behave irradicably or crash.

<u>Resource Constraints</u>

- Not enough time;
- Not enough money.

Testing Implications: Under resource constraints, the tester should meet with developers and users to determine the priority of the different aspects of the system. The tester should test the highest priority items and explicitly determine the acceptable testing levels on each item.

<u>Intended Use</u>

- Distribution of system;
- Mission criticality;
- Cost of an error;
- Importance of the system to the organization.

Testing Implications: The testers should work with developers to assess the impact of the system in terms of the cost of an error, its distribution, and its criticality. If the system is performing a mission-critical function, the cost of an error is high, and the system is to be widely distributed, all aspects of the system must be thoroughly tested. If for any reason this is not possible, the system should not be allowed to become operational.

LESSONS LEARNED/RECOMMENDATIONS FOR TESTING

- Comment the rules, indicate their sources.
- Partition large rules bases into meaningful rule subsets.
- Withhold a portion of test cases from the development effort; reserve some test cases for use during system testing.
- Mix test case development with knowledge engineering; if it is difficult to get an expert's time, use his or her time wisely when one has it.
- Determine the intended users of the system; they may be different from the experts. Test the system with the expert as well as with the intended user.

- Testers must talk to the developers. Ask the developers which areas of the software they feel are strong and which they feel are weak. Concentrate testing on weak areas.
- Explicitly define implicit performance requirements by meeting with developers, experts, and users.

Generally, most of the recent work in the Army has been with prototypes. (The Army's approach to AI has been described as "out with a hammer in search of a nail.") The establishment of Army AI Centers seems to indicate an organizational commitment to go beyond "studying the problem" into developing operational systems. Developers need to allocate time and money for documentation and testing; both aspects have taken a back seat to building the system. Aside from the inherent difficulties with testing knowledge-based systems, the lack of documented requirements, the unavailability of the experts, the lack of tools for static analysis, and the lack of information in testing methods and procedures have added to the difficulty.

APPENDIX A:

POINTS-OF-CONTACT

COMMERCIAL			
Bob Shore	Booz, Allen, and Hamilton		
AGENCY: ACE			
John Benton	Engineering Topographic Lab	(202) 355-2717	*
Dr. Victor LaGarde	Chief of Engineers	(601) 634-2683	*
AGENCY: AMC			
Richard Camden	HEL, Aberdeen Proving Ground		
Mort Hirschburg	BRL, Aberdeen Proving Ground		
Paul Janusz	U.S. Army AMCCOM	(201) 724-4849	
Mike Gedeon	TACOM	(213) 574-6150	
Phil Emmereman	Harry Diamond Labs	(301) 394-3000	
Larry McKowsky	US Army Belvoir RDE Center	(703) 664-2037	
Bruce Thompson	Aviation Applied Technology Directorate	(804) 878-5620	
Lucille Newman	Headquarters AMC	(703) 274-8952	
Dr. Willard Holmes	Missile Command	(205) 876-1048	*
Edward Beach	Communications-Electronic	(201) 544-2176	*
Admiral Piper	Training Aids and Devices	(407) 380-4287	*
Wayne Ammodt	AMCCOM (Chemical Group)	AV 743-5931	*
Jeff Murter	APG Combat System Test Activity	(310) 278-7727	*
Doug Chubb	CECOM (Vint Hill)		*
Rusty Warren	Natick Labs	(313) 574-6150	*
Gerry Coonan	Aviation Systems Command	(314) 263-1955	*
Robert Bell	Test and Evaluation Command Headquarters	(301) 278-7882	*
Susan O'Donnel	Central Systems Design Agency	(314) 263-5045	*
Som Karemchetty	Laboratory Command - HQ	(202) 394-3000	*
David Hislop	Army Research Office	(919)	*
AGENCY: ARI			
Dr. Joseph Psotka	U.S. Army Research Institute for Behavioral Sciences	(202) 274-5540	
AGENCY: CAA			
LTC Michael Ryan	Concepts Analysis Agency	(301) 295-5229	*
AGENCY: DA			
CPT Shawn Butler	HQDA: AI Center	(202) 694-6900	
Dr. Raymond Freeman	Deputy Chief of Staff for Operations & Plans	(202) 556-2942	*
MAJ Chip McConville	Deputy Chief of Staff for Intelligence	(703) 671-8680	*

AGENCY: DARPA			
LTC Peter Sowa	Defense Advanced Research Projects	(202) 695-5918	*
AGENCY: DLA			
Lt. Col Bart Hodgson	DLA	(703) 274-7227	
Larry Johnson	Headquarters, DLA		
Rex McHail	DLA-OSS		
Fred Murphy	DFSC-LO	(202) 274-7448	
John Bryant	Defense General Supply Center	(804) 275	
AGENCY: FORSCOM			
Bob Edmonds	Forces Command	(404) 752-4114	
AGENCY: ISC			
MAJ George Thurmond	Army Institute for Research in Management Information and Computer Science	(404) 894-3110	*
AGENCY: TRADOC			
LTC John James	Headquarters TRADOC	(804) 727-3945	
Ollie Hedgepeth	Logistics Center, Ft. Lee	(804) 734-1621	
Major Randy Ball	Intelligence Center School	(602) 538-2253	*
CPT Patrick Vye	Combined Arms Center	(913) 684-5607	*
CPT Gary Krzisnik	Armor Center/School	(502) 624-6347	*
CPT Keith Roberson	Signal Center/School	(404) 791-6520	*
CPT Don Wilkins	Field Artillery Center/School	(405) 351-6400	*
AGENCY: TSG			
Glen Higbee	USAMARIID	663-7514	
AGENCY: USMA			
Rob Rayenga	West Point Military Academy	(914) 938-3427	
MAJ Robert Richbourg	Deputy Chief of Staff for Personnel	(914) 938-2407	*

*Limited information

APPENDIX B:

SELECTED INTERVIEWS

Interviewee: Richard Camden and Carolyn Dunmyer
HEL, Aberdeen Proving Ground
February 1989

Description: The expert systems were clerical in nature. The systems are generally problem-oriented, i.e., they solved a problem previously performed by a person using judgment and experience. The developers had access to an expert system shell (Knowledge Craft) and used LISP for applications development. The developer viewed LISP as a rapid prototyping language and thought the software would have to be converted to another language (e.g., C) before it was fielded. The systems did not use uncertainty or deep knowledge. They did not yet have experience in testing expert systems. The developer did not see a difference between testing a knowledge-based system and testing conventional software. Testing a knowledge base seemed no different from testing a database.

Requirements: Typically they work from a requirements and design document. The requirements and design documents are located on the same computer on which the software is being developed in order to facilitate updates. One difference that they saw between conventional software and AI software was the frequency with which requirements changed.

Testing: Generally, their approach to testing was to test the software to see if it did what it was intended to do. The approach to testing a knowledge-based system was similar to the approach used to test a decision support system: (1) test the algorithms; (2) determine whether or not the system is inferencing correctly; and (3) validate the inference pathways (the process as well as the answers). They viewed the availability of an expert as essential for properly testing an expert system. They did not see a need for separately testing aspects of the shell.

Testing Tools: They have not used any testing tools. They mentioned a testing tool called CGI that is being developed for Ford. CGI is a tool used to validate knowledge bases, but they did not have access to this tool or any others for testing a knowledge base.

Interviewee: Richard Kaste (Researcher/Systems Developer)
BRL, Aberdeen Proving Ground
February 1989

Description: The expert system was designed for use by combat developers at Ft. Sill. The system is viewed as an aid for decision making rather than as an actual decision maker. The system did not use uncertainty or deep knowledge. The system was in its early stages of development. One critical aspect of the system was its need to interface with many other systems. LISP was used as the development language. No expert system shells were used. The knowledge base consisted primarily of rules that were based on doctrine.

Requirements: The requirements for the system evolved from an existing program. A requirements document did not exist. The requirements for the system were determined from meetings with the Combat Development personnel from Ft. Sill.

Testing: Testing the system consisted of testing the knowledge base by examining the rule base. The rule base was small, consisting of fewer than 100 rules. The rules were tested for consistency and redundancy; they looked for rules which could be collapsed. The current test strategy was basically to change inputs, get outputs, and check if outputs are reasonable (as determined by a Sgt 1st Class based on his experience as an artillery man and his knowledge of doctrine). Testing did not focus on what causes the system to crash since it was being developed for expert users (NCOs). The user interface was tested by getting feedback on the displays from the intended users. The system was stress-tested to see the number of targets that it could handle and the speed at which it could handle them. For this system, speed was an important attribute.

Additionally, several subjective measures were obtained. The developer/tester looked to see if the operator was happy with the results produced by the system, if the system fit in the intended operational environment, the timeliness of the results, and the quality of the explanation facility.

Test cases were derived automatically from a scenario driver developed by a contractor. The scenario driver provided realistic test cases but did not provide cases that would go through all the possible ranges of values for the variables.

One of the problems associated with testing the expert system was that it was difficult to separately test the inference engine and the knowledge base. It was difficult to determine when to stop testing. Quickly changing code and rapidly changing requirements added to the difficulty in testing the expert system. Since no single expert in this area existed and no "one" correct answer existed, it was difficult to measure how well the system was doing. Additionally, it was difficult to get agreement from the experts.

Interviewee: Mort Hirschburg, ALBM
BRL Aberdeen Proving Ground
February 1989

Description: A small (5%) of combat knowledge is in ALBM (AirLand Battle Management), one offensive posture and one defensive posture. The system is very large—on the outer limits as far as size of expert systems. The system contains over 400 procedures, each containing large rule bases. Initially the developers used a commercial shell but, because of run-time performance requirements, it could not be used.

ALBM contains a custom-built series of support tools (graphical, representational, database, mini-operating system, and expert system). These support tools have the potential to become commercial products.

The development language is LISP. When the system is a "final product," it will probably need to be converted to C, then to Ada, which could be a very big problem. The system is considered a "non-toy fragile AI system."

Requirements: The system has a formal requirements document that has been subjected to a formal design review. Feedback was obtained on the initial system design.

Testing: A formal and comprehensive test plan was developed for ALBM. The approach to testing for this system is multi-faceted; it includes static testing, dynamic testing, multiattribute analysis, acceptance by experts, and questionnaires. The multi-faceted approach consists of obtaining feedback from outside experts as well as developer experts. Additionally, a customized application of EVA is being developed for static analysis of the knowledge base.

Currently, changes to the knowledge base in the field are not addressed. The goal is to have a computer-literate person to change ALBM—not a novice user. The idea is that staff officers can tailor parts of the system to the commander's preferences. He was unsure on how to comprehensively test a system with this kind of flexibility. It is likely that pieces of the system—rather than the entire system—will be fielded, since the pieces may be more stable.

Additionally, he indicated that there were special problems regarding testing classified systems, such as the availability of good test data and providing the necessary safeguards for those who need to access the system to test it.

Interviewee: Glen Higbee and Tim Cannon
USAMARIID, Ft. Detrick, MD
February 1989

Description: The department at USAMARIID provides computer support for research—medical research (vaccines, drugs, early detection of disease, treatments, defensive biological warfare) and basic research in chemistry, biology, and micro-biology.

The development environment is informal. The software is typically developed with an expert system shell (KEE, Level 5, 1st Class, XPER) or in LISP. The applications are primarily designed for use on a PC. All systems are used in-house, characterized by trained users, and the users have easy access to developers. Because of this work environment, the software developers are able to get feedback from the "expert" users as to how the system performs in actual settings and make modifications on request. Some examples of the expert system applications are for use in drug and vaccine development and for systems that are used as a recruitment training aid for selection of volunteers for human vaccine testing, immunization scheduling, and small animal breeding. Other expert system applications have been developed for disease prediction and mosquito classification.

Requirements: Typically there are no documented requirements. The software requirements are determined from meetings with the researcher and feedback on the prototype as it is developed. The development schedule for software is mission-related (software in support of highest priority mission is given highest priority for development).

Testing: No formal test plans are developed. Typically a single expert is used to test the system. The developers believed that the expert system could not be developed or tested without a commitment from management to make that expert available. Generally the software is not tested with or used by novices. The error-handling routines that flag data outside acceptable bounds are tested. A limited amount of testing is performed on the interface by the developer. Users exercise the interface when software is in actual use; because of the close working relationship between the end user and developer, the developer is able to obtain feedback and make the appropriate adjustments.

Test cases did not present a problem; actual data are readily available for use in testing. Initially a subset of actual data is used for testing, then the software is tested against the entire database. Generally, the testing strategy here is to test the system as it is developed, testing it first with a large subset of an actual database and then testing it with the entire database. The system is tested with the expert throughout development and is monitored as it is used.

One of the biggest problems they saw in developing and testing AI systems is the lack of a standard development methodology and an unclearly defined flow of control.

Interviewee: Paul Janusz
Picatinny Arsenal
February 1989

Description: One expert system project was initiated in 1983/84; it was a Phase 1 SBIR to apply AI to SQA (an expert system to do documentation reviews, test plans, and statements of work). The purpose of Phase 1 was to examine the feasibility of such an expert system. The developers (a contractor) used an expert system shell called Invisage (by Systems Designers International). They experienced many problems with using the shell: the shell contained bugs, the capabilities and limitations of the shell were not fully understood, there were no training courses on using the shell, and there was a lack of customer support from the maker of the shell. Additionally, the shell was designed for a VMS operating system and everyone in the organization used a UNIX operating system.

Requirements: No documented requirements existed for the system. The entire development effort lacked focus on a specific domain area; the problem was not well-defined.

Testing: A limited amount of testing was performed. The system was never used because it did not run under a UNIX operating environment. No static tests were performed on the knowledge base. Initially one expert was available for testing but, because of a high turnover rate, was unavailable for testing the system.

Other: Some of the lessons learned from their initial efforts were to get a better idea of the system requirements up-front and to obtain a commitment from management to make the expert available for developing and testing the system.

There are now plans to implement a Phase 2 version of the software. The new effort will be programmed in C and will use the NEXPERT expert system shell. With the new effort, the development team plans to integrate testing with all phases of development. It will require a test procedure up-front and implement a series of acceptance tests along the development path.

Interviewee: Larry McKowsky
Fort Belvoir, RD&E Center
March 1989

Description: The Center develops knowledge-based systems for in-house use. The expert system described was a statement of work generator. Before development, they looked at shells that could be purchased for under 5K, and selected XSYS. It was the first time the developers used the shell and many problems were associated with it. To this group, copyright/licensing presented a big problem. The restrictions made it difficult and costly to share the software and limited the potential end users.

Generally, people were disappointed with the results of the development effort and the software was described as "more fizz than sizzle." They were hoping the system could function as a tutorial for new employees in the organization, but instead, the software ended up as an automated "checklist." The system was described as capable of providing the user "one year of experience twenty times."

Most of the rules in the knowledge base came from regulations; when the regulations became outdated, the system became virtually useless.

Requirements: A requirements document for this system did not exist.

Testing: No testing was specifically aimed at the knowledge base and no automated static analysis tools were used. Generally, testing consisted of the expert using the software and determining if the results were reasonable. A problem faced in testing this software was testing the shell. It seemed the shell contained many bugs which made it difficult to pinpoint the reason for an error occurring. Another difficulty cited was the lack of a standard to measure the success of the system.

Other: The lessons learned from their initial experience in expert system development were specifically aimed at issues concerning the expert system shell. They advised allocating time to learn the language of the shell and to gather information on the shell (its limitations and capabilities, additional costs associated with site licenses, and vendor support).

Interviewee: Larry McKowsky and Lydia Carrasquillo
Fort Belvoir, RD&E Center
March 1989

Description: The system used the TI Personal Consultant. The system's knowledge base was developed based on interviews with experts who were not always willing to participate. A prototype was developed by a contractor and was tested in-house for a two-week period.

Testing: The approach to testing involved examining three aspects of the system—technical, operation, and performance. Initially, the system's output was reviewed by an expert. The expert felt that, although the logical sequence of questions presented seemed correct, the output was incorrect. Additionally, the system was too slow. After these general findings, the knowledge base was examined in detail. First, the automated rule checker that came with the shell was used to examine the knowledge base, but did not thoroughly exercise the knowledge base.

The testers tried to test and verify every rule, which meant manually examining every rule. As a result, missing rules, rules that were not fired, and rules that could be combined were identified. The testers were able to collapse the rule base—consisting of 500 rules—to 150 rules. Additionally, the testers checked the logic used by the system by going back to the experts and asking them to verify some of the work performed by knowledge engineers. The logic was verified by interviews with multiple experts. The testers expressed difficulty in reaching agreement with the multiple experts.

Other: One of the greatest difficulties they faced was gathering knowledge for the knowledge base. The number of changes makes the expert system extremely difficult to upkeep. The developers/testers believed it was important to consider maintenance issues up-front and determine a means to keep a knowledge base current with Army regulations. They were also unsure of the ability of an expert system to teach a nonexpert to become an expert in a specific domain area.

Some of the lessons learned follow:

- Experts may not exist or may be difficult to find.
- Define clearly and early what you want the system to do.
- Think about maintenance early in the development process.
- Testing should be ongoing throughout the development process.

- Do things small, make sure you have access to relevant knowledge.
- Pick a subject area that you know something about.
- Keep the scope small.
- Pick something that people in the organization really care about.

Interviewee: Lt. Col Bart Hodgson
Defense Logistics Agency
March 1989

Description: Two prototypes resulted from two years of exploratory involvement with expert systems. The Buyer's Assistant puts together clauses for contracts in purchasing fuels. The Buyer's Assistant was developed with CLIPS for use on a Gould 950. The Inventory Manager's Assistant (IMA) monitors inventory demand and procurement, specifies re-order points, and reviews information. The system flags information that needs to be updated. The Inventory Manager's Assistant was developed using the M.1 expert system shell and was initially intended to be used on a PC, but migrated to a mini-computer. Both systems had a critical need to access large central databases. The development of these systems was an iterative process of building a prototype, showing it to an expert, and obtaining feedback.

Requirements: A requirements document did not exist for the prototypes. The prototypes were used for generating statements of work for the operational system.

Testing: The testing strategy is to develop a comprehensive set of test cases. The test cases are analyzed by a panel of experts to determine the expected outcome. The computer system is used to determine the output for the same set of cases. The experts and the expert system should reach the same conclusion 85% of the time.

Other: Last year DLA had put together a three-person working group to develop a management strategy on how to do AI and implemented an AI coordinating group and a training program.

Some of the lessons learned follow:

- For the expert system development process to be successful, management must commit to making the expert's time available.
- Develop in-house capabilities. Implement training programs in knowledge engineering as well as in the expert system shell.
- Instill order. Identify a set of AI products to use in the agency and provide training.
- Require software developers to develop software that is consistent across machines and to use similar interfaces across machines.

- Design software with the capability to move up to a bigger machine.

- System Integration is very important. Machines and access to very large databases may drive requirements.

For DLA, expert systems must, regardless of size, be able to access data. Additionally, the expert systems should be incorporated into "modules" that can be accessed from programs written in other languages.

The biggest stumbling block faced by this organization was trying to get an organizational commitment to go beyond just "studying the problem."

Interviewee: Shaun Butler
HQDA AI Center, Pentagon
May 1989

Description: Currently work in AI is of two types—knowledge-based systems and object-oriented programming. The iterative development process is characterized by rapid prototyping. One expert system, soon to be fielded, captured the expertise of a single psychiatrist. The system determines a benefit rating for people unfit for service due to a mental disability. Basically the system is a classification system designed for use by nonexperts.

Requirements: A requirements document did not exist for initial efforts. There are plans to convert the system from KEE on a Symbolics to ART for a PC, and there are plans to document the requirements for the conversion.

Testing: The approach to testing involved a single expert as well as tests with the end users. A tool for static analysis was not available and most of the examination of the rule base was performed manually. When testing the knowledge base, they tried to find redundant rules and assess the effects of different rule combinations and adding rules to the knowledge base.

A comprehensive set of 200 test cases was developed. The cases covered the different types of disorders as well as variability within a particular type of disorder. The test cases attempted to approximate the population and were obtained from actual data.

The expert used the machine to solve the 200 test cases. The expert used the patient's mental health summary data as input. All of the expert's input and keystrokes were saved and analyzed at a later time. (They also built tools that used the saved input data to rerun the system after changes were made to the rule base.) This testing stopped when the system and the expert agreed on the answer 80% of the time.

The system was also tested with the end users. The system was intended for non-psychiatrists that serve on a board to determine the benefit compensation rate. Boards generally consist of a doctor (of some type), a line officer, and an administrative person. The system was designed to take away some of the bias and discrepancies among different boards. The system was tested by board members. One board representative input values and these values were saved and compared to the expert's interpretation of the same patient's data.

Other: The developers found it difficult to obtain a documented comprehensive testing strategy in the literature. There was a need for a method to test reasonable combinations of rules since it was not always possible to go through every rule in the rule base. Additionally, they found it difficult to determine a measure of performance and ways to determine when the expert was wrong.

Because of resource constraints, the same 200 test cases were used in both developing and testing the system.

Interviewee: John Bryant/Jim Anderson
Defense General Supply Center
May 1989

Description: One system developed by the Defense General Supply Center was a hazardous material classification system. The system was designed for use by clerks at depots who receive hazardous materials and who must consistently and accurately assign one of fifty-five hazardous material codes to a shipment so that a storage location can be assigned. The system was designed to minimize the number of user input errors. Generally, the system prompted users to select an item from a menu rather than enter a response using the keyboard. The prototype was developed for use on a PC, used 94K, and contained 400 rules. The prototype assigned 5 corrosive codes, 3 flammable codes, and 2 corrosive and flammable codes. The prototype was developed with the M.1 expert system shell. The knowledge base incorporated rules from regulations (Department of Transportation, United Nations, and Maritime).

Requirements: The software was developed in-house as a prototype to assess the feasibility of a full system; no requirements were documented.

Testing: A group of experts—chemical engineers—were used to test the system. The chemical engineers were given the latest version of the prototype and maintained a very detailed spreadsheet of things that went wrong. The chemical engineers also made sure the system was consistent. The system was not tested with the end user.

Other: It was very difficult to get the chemical engineers to agree with each other as to the correct code.

One particular problem in testing knowledge-based systems was in finding rules that arrive at the right conclusion for the wrong reason. Other questions were raised as to what constitutes an expert. Knowledge bases incorporate rules from various sources, such as regulations and directives as well as subject-area experts.

The current approach to AI in the Army was described as "out with a hammer in search of a nail when what you really need is a screw." Organizationally, there have been a lot of people jumping on the AI bandwagon and many people have expected more than AI could deliver. The interviewees saw a need for more education as to what is AI, what are appropriate applications, and a practical way for them to be implemented.

Some of the lessons learned which they described include:

- It is easy to underestimate the amount of resources it will take to complete the job.
- Anticipate false starts.
- Make sure the problem is relatively small and well-defined.
- Assess the availability of actual data early.

Interviewee: Dr. Joseph Psotka
Army Research Institute
July 1989

Description: The system is an intelligent training system designed for classroom use at the air defense school. The system is very large and designed to run on a Symbolics. The software was developed using KREME, an object-oriented shell. The system was designed to improve the quality of training, as well as reduce training costs. Previously, students trained with an actual radar system that cost approximately 3 million dollars. The intelligent training system simulates the HAWK air defense radar to reduce the cost of training.

Multiple experts were used in development. Three types of experts were used (engineers, trouble-shooters, and experts with HAWK radar) and four different levels of expertise were incorporated. The user interface and explanation facility were viewed as the most critical aspects of this system.

Requirements: No formal requirements documented existed for this large system, but the design specifications were documented. The air defense school did require it to fit into the school's curriculum.

Testing: As it was designed, Raytheon (the contractor) and the air defense school used the system directly or looked at the design specifications to assess the completeness and quality of radar simulation. The interface was iteratively evaluated by the end users (students and trainers). The interface evaluation included an assessment of the screen design, feedback message placement, scrolling, features, menu naming, design, and actions. The system was evaluated in a classroom setting by observing the system in use and through questionnaires. The questionnaires addressed features used, perceived usefulness, perceived problems, and general feelings regarding the system. Observers videotaped and took notes to assess the usefulness of the system in a classroom setting. Additionally, an experiment using subjects in an actual classroom setting was designed to measure the impact of the system on student performance.

Other: The system is now operational and part of the instruction program at Ft. Bliss.

Interviewee: LTC John James
Director, TRADOC AI Center
July 1989

Description: The TRADOC AI Center has developed one very large system and several small systems. The small systems include a cost estimator and a media selection expert system. The cost estimator was developed in one month by a West Point cadet on a work-study program. The program develops cost estimation relationships for allocating TRADOC dollar resources. The software was developed for use on a Macintosh PC and was written in Hypercard. The system is used by the resource management office and performs work in 5 minutes that previously took about 2 weeks. The other small system, an expert system for training media selection, is a PC-based system written in CLIPS. The system was designed to help select an appropriate training medium based on the level of expertise a student is expected to obtain (familiar, conversant, or mastery). The system will be distributed to all TRADOC schools.

The large system—the Inventory Asset Analyzer—is an object-oriented program developed in LISP to run on a Symbolics. The Inventory Asset Analyzer develops a modernization plan for the Army's fleet of rotary wing aircraft from a given set of assumptions, distributes resources to the highest priority units, and redistributes the usable displaced resources to lower priority units. The system also provides extensive "what if" analysis. The software has introduced a substantial time and cost savings and is being enhanced to deal with new modernization issues to show the logistical, maintenance, and life-cycle costs of future aircraft modernization plans.

Requirements: The systems were developed by rapid prototyping and no system requirements were documented.

Testing: There were no formal test plans for these systems. Testing consisted of iteratively running the system with the expert and potential user.

Other: Some of the lessons learned include:

- It is important to select the right problem and to match the AI technology with the problem.

- Rapid prototyping has worked very well for the TRADOC AI Center; it has cut down on long development cycles for software.

- Mix small and large projects, get the staff involved in small- and medium-sized projects, and develop training programs.

- To successfully develop AI systems, both the expert and user need to be involved.

- For developing successful applications of expert systems, TRADOC recommends selecting the right problem, using qualified people, trying not to oversell the technology, committing the necessary resources, clearly focusing the problem, and using shells, particularly for smaller applications.

Interviewee: Ollie Hedgepeth
Logistics Center, Ft. Lee
July 1989

Description: The AI branch at the Logistics Center at Ft. Lee generally limits the development of AI systems to one large application and several smaller applications. The small systems are generally designed with a specific expert user in mind. Generally, the systems seem to be smart front or back ends to existing systems. One of the small systems is the Study Plan Advisor. The Study Plan Advisor helps to produce a study plan in accordance with Army Regulations (AR 5-5, *Management of Army Studies and Analysis*). The large system—Planning Assistant for Logistics Systems (PALOS)—was developed on a LISP machine and was designed for developers of Standard Army Management Information Systems (STAMIS). The system shows the relationship of Army units and their STAMIS requirements. PALOS also allows for extensive "what if" analysis. PALOS has reduced the amount of time for the analysis from a 6-month manual process to a 2-hour automated process.

Requirements: Generally, system requirements have not been documented.

Testing: Generally, testing involves developing a set of test cases, using the system to solve the test cases, and comparing the system results to the results obtained when the problem was solved by an expert. Additionally, they will ask a nonexpert (such as a new person) to solve the problem with the system and compare the nonexpert's results to the expert's results.

Additionally, some testing was performed on the knowledge base, but it was done manually since they did not have any automated tools for static analysis.

Other: Some of the lessons learned include:

- Ask early how the system will be maintained.

- Get the users involved early in the development process.

- It is important for the AI technology to be accepted by an organization. Acceptance is necessary to obtain resource commitments from management. Small successes (with fast turnaround times) often pave the way for larger systems.

- For the Army AI Center to work effectively, a mix of military and civilians is needed, since solders will be reassigned. The exception to this seems to be when there is a large group (6) of military, such as at the AI Center at the Pentagon or at TRADOC.

- To make the system acceptable to the domain expert, quantify the improved performance in the number of hours saved by using the computer system.

Interviewee: Brian Thompson
AVSCOM, Aviation Applied Technology Division
July 1989

Description: The expert systems developed were diagnostic systems for helicopter repair. The diagnostic systems were designed to be used in the field and run on portable computers. The systems involved extensive rule bases with data gathered from expert troubleshooters.

Requirements: Documented requirements did exist for these systems.

Testing: The prototypes were tested using actual cases from the past. Initially, the systems were tested in a similar operational environment for 3 months. At this time, the overall system effectiveness and the user interface were evaluated and changes were made to the system. The system was then run parallel to the existing process in the intended operational environment for approximately one year. During the parallel test, assessments were made as to how well the system met the goals stated in the requirements document.

Other: The experiences with the helicopter diagnostic system highlighted the importance of adequate field testing. The field testing uncovered many unexpected errors.

Other lessons learned from their experiences are as follows.

- It is important to comment the rules. Rules which were written to increase speed were very difficult to understand, while rules which were easy to understand took longer to process.

- It is useful to partition the rules into "rule sets;" it increases program speed and allows for more manageable debugging.

- It is essential to have a strong commitment from management for the project to be successful.

APPENDIX C:

SURVEY OF ARTIFICIAL INTELLIGENCE IN THE ARMY

SURVEY CONDUCTED

1. General Information:

Name:

Address:

Phone:

Office Background (Please answer for your office):

Number of people or level of effort in AI: Civilian
Military

Amount of contractor effort in AI (persons or dollars):

Number of years the office has been working in AI.

2. Development Environment:

Level of effort involved in the systems your office creates. (For each category of effort, enter one, some or many.)

	EFFORT (MAN-MONTHS)				
	<1	1-6	7-24	25-60	>60
HOW MANY: (one/some/many)					

How many of your office's AI systems are on the following hardware platforms (enter one, some, or many)?

Micro-computer:

Workstation:

Mini/Mainframe:

How many of your office's AI systems utilize an AI shell or environment such as KEE, CLIPS, M-1, Exsys, or 1st Class (one/some/many)?

Which shells or environments?

How many of your office's AI systems utilize an AI language such as LISP or Prolog (one/some/many)?

Which languages?

Will the final version of the AI systems be in a shell, an AI language, or a conventional language (e.g., Ada, Basic, C)?

What types of AI systems has your office created (one/some/many for each type)?

Expert system or knowledge-based system:

Image recognition:

Intelligent tutoring:

Machine learning:

Natural language:

Neural network:

Other (please specify):

3. How many of your office's AI systems would you classify as (note that categories are not mutually exclusive):

CLASSIFICATION:	MISSION-CRITICAL SYSTEMS	SYSTEMS TO ASSIST AN EXPERT	SYSTEMS TO ASSIST A NOVICE	SYSTEMS THAT OPERATE AUTONOMOUSLY
HOW MANY: (one/some/many)				

How many will be used by the organization that developed them (one/some/many)?

How many will be distributed widely (one/some/many)?

4. We have noticed eight common methods for testing. How many of your expert systems have been tested using something similar to each of these approaches (see last page for definitions)? (Note: one system may be tested by multiple approaches.)

APPROACH:	NO TESTING	PROTOTYPE FOREVER	AGREE-MENT	COMPLI-ANCE	SATIS-FACTION	CASE-DEPENDENT	ORGANIZA-TIONAL	FIELD TESTING	MULTI-FACETED	OTHER (Please explain)
HOW MANY: (one/some/many)										

5. Factors that Affect Testing:

To what extent have the following factors affected the quantity and quality of testing performed on your office's AI systems?

	LITTLE				A LOT
INFORMATION AVAILABLE ON TESTING	1	2	3	4	5
TIME	1	2	3	4	5
RESOURCES	1	2	3	4	5
DEVELOPMENT ENVIRONMENT	1	2	3	4	5

6. In your office's testing, how often are the following aspects of an expert system or knowledge-based system tested (one/some/many)? How important is each aspect (low/medium/high)?

Structure and content of the knowledge base (consistency, completeness, and accuracy):

Inference engine:

"Service requirements" (compatibility with host computer system, computer usage, system integration, portability):

Performance (quality of results, speed):

Usability (ease of use, extent of use, adequacy of interface, organizational impact):

Other (please explain):

7. Software Quality:

For any AI system, how important is each of the following traditional software quality factors (low/medium/high)?

Efficiency (How well does it utilize resources?):

Integrity (How secure is it?):

Reliability (What confidence can be place in what it does?):

Usability (How easy is it to use?):

Correctness (How well does it conform to the requirements?):

Maintainability (How easy is it to repair?):

Testability (How easy is it to verify its performance?):

Flexibility (How easy is it to change?):

Interoperability (How easy is it to interface with another system?):

Portability (How easy is it to transport?):

Reusability (How easy is it to convert for use in another application?):

8. Future Work:

How important is future work in these areas (low/medium/high)?

Develop methods for testing the validity of "confidence" or "uncertainty" factors, which are used in some expert systems:

Develop methods and aids for determining the accuracy and built-in bias of an expert system:

Develop automated static testing tools for analyzing the consistency and completeness of knowledge bases:

Develop an automated means for assessing user acceptance:

Develop a tool to generate requirements against which an AI system can be tested:

Develop a computer program to help perform multi-faceted testing:

Try out AI testing methods in a "testbed:"

- Would your office like to serve as a testbed (yes/no/maybe)?

Periodically update and publish a compendium of lessons learned from efforts to test Army AI systems:

Collect and publish an anthology of articles on testing AI:

9. Any Other Testing Considerations:

10. Comments on Questionnaire:

Did this questionnaire raise new testing issues (yes/no)?

Would you like to receive a copy of the results?

Do you feel a questionnaire of this nature should be periodic (e.g., make this The Annual Army AI Census)?

TABULATION OF SURVEY RESULTS

Level of effort involved in the systems created:

	EFFORT (MAN-MONTHS)				
	<1	1-6	7-24	25-60	>60
ONE	1	0	2	0	0
SOME	1	2	3	2	0
MANY	1	5	2	1	1

How many AI systems are on the following hardware platforms?

	MICRO-COMPUTER	WORKSTATION	MINI-MAINFRAME
ONE	1	1	0
SOME	5	5	2
MANY	4	2	0

How many AI systems use a shell or an AI language?

	SHELL	# OF AI LANGUAGES
ONE	0	2
SOME	3	1
MANY	4	6

What types of AI systems have been created?

	EXPERT SYSTEM	IMAGE RECOGNITION	INTELLIGENT TUTORING	MACHINE LEARNING	NATURAL LANGUAGE	NEURAL NETWORK
ONE	1	1	0	2	0	2
SOME	2	0	1	1	0	1
MANY	7	0	0	0	1	0

How many AI systems developed would be classified as:

	MISSION-CRITICAL SYSTEMS	SYSTEMS TO ASSIST AN EXPERT	SYSTEMS TO ASSIST A NOVICE	SYSTEMS THAT OPERATE AUTONOMOUSLY
ONE	0	1	1	0
SOME	1	3	4	2
MANY	3	4	2	0

How will the AI system be distributed?

	USED BY DEVELOPING ORGANIZATION	WIDELY DISTRIBUTED
ONE	2	3
SOME	4	7
MANY	1	2

How many expert systems have been tested using something similar to these approaches?

	NO TESTING	PROTOTYPE FOREVER	AGREE-MENT	COMPLI-ANCE	SATIS-FACTION	CASE-DEPENDENT	ORGANIZA-TIONAL	FIELD TESTING	MULTI-FACETED
ONE	0	1	1	0	0	0	0	1	0
SOME	1	1	1	2	3	0	0	2	1
MANY	0	3	2	1	1	1	1	2	1

To what extent have the following factors affected the quantity and quality of testing performed?

	LITTLE 1	2	3	4	A LOT 5
INFORMATION AVAILABLE ON TESTING	1	3	4	0	1
TIME	0	1	2	1	5
RESOURCES	0	1	3	2	3
DEVELOPMENT ENVIRONMENT	2	2	2	3	0

How often are the following aspects on an expert system or knowledge-based system tested?

	KNOWLEDGE BASE	INFERENCE ENGINE	SERVICE REQUIREMENTS	PERFORMANCE	USABILITY
ONE	0	0	1	0	0
SOME	4	2	2	2	3
MANY	4	3	2	7	5

How important are the following aspects of expert system testing?

	KNOWLEDGE BASE	INFERENCE ENGINE	SERVICE REQUIREMENTS	PERFORMANCE	USABILITY
LOW	0	0	0	0	0
MEDIUM	2	3	1	2	0
HIGH	4	0	2	4	1

How important is each of the following software quality factors?

	HIGH	MEDIUM	LOW
EFFICIENCY	3	5	2
INTEGRITY	3	2	5
RELIABILITY	8	2	0
USABILITY	10	0	0
CORRECTNESS	7	3	0
MAINTAINABILITY	2	7	1
TESTABILITY	5	3	2
FLEXIBILITY	5	4	0
INTEROPERABILITY	4	2	4
PORTABILITY	3	5	2
REUSABILITY	2	3	5

How important is future work in the following areas?

	HIGH	MEDIUM	LOW
METHODS FOR TESTING THE VALIDITY OF CONFIDENCE OR UNCERTAINTY FACTORS	1	4	4
METHODS AND AIDS FOR DETERMINING THE ACCURACY AND BUILT-IN BIAS OF AN EXPERT SYSTEM	3	3	2
AUTOMATED STATIC TESTING TOOLS FOR ANALYZING THE CONSISTENCY AND COMPLETENESS OF KNOWLEDGE BASES	6	2	0
AUTOMATED MEANS FOR ASSESSING USER ACCEPTANCE	2	2	2
TOOL TO GENERATE REQUIREMENTS AGAINST WHICH AN AI SYSTEM CAN BE TESTED	1	3	5
COMPUTER PROGRAM TO HELP PERFORM MULTI-FACETED TESTING	0	4	4
TRY OUT AI TESTING METHODS IN TESTBED	3	1	1
PERIODICALLY UPDATE AND PUBLISH COMPENDIUM OF LESSONS LEARNED FROM EFFORTS TO TEST ARMY AI SYSTEMS	4	2	3
COLLECT AND PUBLISH ANTHOLOGY OF ARTICLES ON TESTING AI	5	1	4

APPENDIX D:

LESSONS LEARNED

Richard Kaste of the Ballistic Research Laboratory, Aberdeen Proving Ground, provided a list of lessons learned primarily regarding knowled‹,e engineering and development. (The list below was received EMAIL and quo ed from the EMAIL response.)

- The process of building an expert system is inherently experimental.
- Knowledge engineering is an art.
- Expect much rewriting (e.g., of conflicting information).
- The task will take him much time.
- Expect the unexpected.
- Read documents on the subject, especially those written by the expert.
- Learn the domain to ease the conversation.
- Become familiar with the problem before beginning extensive interaction with the expert.
- Use the terms and methods that experts use.
- Define the task very clearly.
- May be problems with unspoken priorities; may not have one best solution.
- Clearly identify and characterize the important aspects of the problem and subproblems.
- Define what a solution looks like, what concepts and processes are used in it, and the essential aspects of human expertise.
- Define the nature and extent of "relevant knowledge" that underlies the human solutions.
- Define situations that are likely to impede solutions.
- Beware the overzealous expert who expands the problem at the expense of a solution to the one at hand.
- Work intensively with a core set of representative problems.

- Aim for simplicity in the inference engine.
- Don't worry about time and space efficiency in the beginning.
- The expert may provide initial objects, bounds, etc.—not just rules.
- Define the given and inferred data and how they are acquired.
- There may be problems with consistency and completeness.
- There may be problems with fuzziness, since the computer requires some bounds.
- Record the conversations.
- Use two interviewers/notetakers to check for misunderstandings.
- May find it helpful to use blackboard while discussing.
- Learn to interview.
- Watch the expert work.
- Record a detailed protocol of the expert solving at least one prototypical case.
- Can use real-time pressure solutions from the experiment as strawmen for developing actual solutions.
- If he can't explain right then, come back to it; let him think about his reasoning if not sure.
- Ask if he always/never does something; may spin off to special cases.
- Restate the expert's rules; ask if this is what he meant.
- Ask many how and why questions.
- Ask what he "really" does; ask what part of the book methods he ignores.
- Ask questions such as, "if we assume this, what will be bad about the reasoning?"
- Look for structure underlying the expert's changing verbalization.
- Don't reveal representation/methodology to the expert.
- Don't necessarily consider any particular representation.
- Most expert talk will fall into if-then, but there will be special kludges or many rules needed to handle certain cases.

- If a rule looks big, it is.
- If several rules are very similar, look for an underlying domain concept.
- If data are indeterminate, it is probably best to go to a higher level, without defaults.
- May not have to set up example situations to avoid rambling.
- Tell the experts ahead of time what the meeting will discuss.
- Talk to the expert ahead of the actual interview.
- Let the expert know ahead of time what the (important) task is; build up his enthusiasm and self-perception.
- Comment the rules, indicating their sources.
- Engage the expert in the challenge of designing a useful tool.
- Give the expert something useful on the way to building a large system.
- Run the system for the expert to make sure it is doing as he does.
- Don't let the expert play directly with a breakable prototype.
- Get the expert to criticize questions and the system.
- Show the expert some results of his knowledge on a prototype—good for self-esteem and may precipitate useful criticism.
- Provide a "gripe" facility.
- Look for one acknowledged expert; maybe one of three will surface as the best, or they will learn from each other.
- Some form of Delphi technique may prove useful.
- In rectifying multiple experts, watch for rivalry and deferring to higher rank.
- Compare and contrast methodologies.
- Validation may be difficult with multiple experts.
- Resolving conflicting techniques with multiple experts may be difficult.
- Identify the intended users of the final system.
- Involve the eventual users in the design.

- Ask early how the expert would evaluate the performance of the system.
- The user interface is crucial to the ultimate acceptance.
- Make I/O appear natural to the user.
- Insulate the expert and the user from technical problems.
- When testing, consider the possibility of errors in I/O characteristics, inference rules, control strategies, and test examples.
- Keep a library of cases presented to the system.
- Ask early how you will evaluate the success of your efforts.